Programando tareas con Scheduled de Spring

Alex Felipe
03/03/2021

Estoy desarrollando un sistema de ventas web utilizando **Spring Boot** y ahora el cliente pidió algunas características un poco peculiares. Él quiere saber cómo es el flujo de ventas durante un período determinado, por ejemplo, cada hora, o cada día. ¿Cómo podemos hacer esta rutina que llevará a cabo estas tareas programadas?.

Utilizando la API Timer de Java

Inicialmente en una implementación utilizando las API disponibles en Java, podríamos hacer uso de la clase `[Timer](https://docs.oracle.com/javase/1.5.0/docs/api/java/util/Timer.html)`:

```
Timer timer = new Timer();
```

A continuación, haríamos uso del método `scheduleAtFixedRate()` que espera una implementación de la clase abstracta `[TimerTask](https://docs.oracle.com/javase/7/docs/api/java/util/TimerTask.html)` como primer parámetro y dos parámetros más del tipo long que indican el inicio de la ejecución y tiempo constante para volver a ejecutar sucesivamente en **milisegundos**:

```
Timer timer = new Timer();
timer.scheduleAtFixedRate(new TimerTask() {
```

```
    @Override public void run() {
    System.out.println("¡Ejecutando la primera vez en " +   "1 segundo y las d
  }, 1000, 5000);
```

En otras palabras, en este ejemplo estaríamos ejecutando inicialmente el código, después de un segundo, y luego, ¡todas las veces después de 5 segundos!.

Teniendo en cuenta este ejemplo, para que podamos realizar esa consulta al cliente que comprueba el flujo de ventas durante un período determinado, como una hora, haríamos algo como:

```
public class VerificadorDePegos {
  private final long SEGUNDO = 1000;
  private final long MINUTO = SEGUNDO * 60;
  private final long HORA = MINUTO * 60;

  public void verificaPorHora() {
    Timer timer = new Timer();
    timer.scheduleAtFixedRate(new TimerTask() {

      @Override public void run() { *// Código que realiza la consulta de fluj

      }, 0, HORA);
    }
  }
```

En otras palabras, ¡podríamos crear una clase encargada de verificar las ventas realizadas cada hora y darnos la respuesta que nuestro cliente está esperando!.

Utilizando la annotation @Scheduled

Sin embargo, ¡mire la cantidad de código que tuvimos que escribir para que fuera posible realizar esta tarea! En este caso, dado que estamos usando Spring, ¿hay una manera más fácil?

¡Afortunadamente tenemos la annotation [@Scheduled](https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/scheduling/annotation/Scheduled.html) que ya desempeña todo el papel del `Timer` y `TimerTask`! En otras palabras, haríamos lo mismo que hicimos anteriormente de la siguiente manera:

```
public class VerificadorDePegos {
  private final long SEGUNDO = 1000;
  private final long MINUTO = SEGUNDO * 60;
  private final long HORA = MINUTO * 60;

  @Scheduled(fixedDelay = HORA)
  public void verificaPorHora() {     *
    // Código que realiza la consulta de flujo de ventas*
    }
}
```

¡Observa que el código ha disminuido mucho! Sin embargo, ¿qué está pasando ahora?. En el momento que usamos la annotation `@Scheduled` le decimos a Spring que ese método será programado.

Además, observa que el parámetro que se está enviando es `FixedDelay`, este parámetro es equivalente al último parámetro del método `scheduleAtFixedRate()` de la clase `Timer`, es decir, ¡es el tiempo fijo que siempre estará corriendo!. También podemos usar otros parámetros si es necesario, aquí hay algunos ejemplos:

- **initialDelay**: Configura la espera inicial de la programación, básicamente es equivalente al primer parámetro del método `scheduleAtFixedRate()`. También recibe valores `long` indicando los milisegundos.
- **fixedRate**: Configura un período fijo entre el inicio y el final de la ejecución de la programación.

Tenga en cuenta que estamos haciendo lo mismo que hicimos antes, ¡pero con un código mucho más sencillo de entender!

Configurando la programación en Spring

Conocemos el Spring como un framework que hace mucho más fácil la vida del desarrollador, es decir, con poca configuración es capaz de gestionar el ciclo de vida de los objetos entre muchas otras cosas.

Sin embargo, para que esto sea posible, al menos una vez, ¡debemos decirle a Spring que se comporte así!. Por lo tanto, para ejecutar la programación de las tareas con Spring, también será necesario realizar algunas configuraciones.

Transformando la clase en Component

La primera es hacer que la clase que esté usando la `@Scheduled` se gestione por Spring. La forma más sencilla de hacer esto es básicamente convertirla en un `[Component](https://docs.spring.io/spring/docs/current/javadoc-api/org/springframework/stereotype/Component.html)` a través de la annotation

```
@Component public class VerificadorDePegos {

  *//Atributos*

  @Scheduled(fixedDelay = HORA)
  public void verificaPorHora() {
    *// Código que realiza la consulta de flujo de ventas*
    }
}
```

Habilitando la programación

Ahora que la clase está siendo gestionada, solo necesitamos informar a Spring que queremos habilitarle la programación a través de la annotation `@EnableScheduling`:

```
@Component @EnableScheduling
public class VerificadorDePagos {

e*//Atributos*

  @Scheduled(fixedDelay = HORA)
  public void verificaPorHora() {
```

```
    *// Código que realiza la consulta de flujo de ventas*
    }
}
```

Listo ¡nuestra clase ya está configurada! ¡Ahora simplemente reinicie su proyecto de Spring Boot y la programación se ejecutará!

Recordando que para realizar una prueba más notoria utilice la base de los segundos en el parámetro `fixedDelay` o `fixedRate`. ;)

Programación por un determinado horario del día

Observa que ahora conseguimos solucionar la situación en la que el cliente quería ver la cantidad de ventas cada 1 hora, es decir, hicimos una programación cada X tiempo. Sin embargo, si el cliente llega y dice lo siguiente:

"¡También quiero un informe que me muestre cuántas ventas se realizaron hasta el mediodía todos los días!"

¿Y ahora?, ¿cómo podríamos hacerlo?. Bueno, una solución inicial sería que nuestra tarea comenzara a las 12:00, y luego, tendría un tiempo fijo de 24:00.

Sin embargo, observa que este tipo de solución no es adecuada, después de todo, tendremos algunas complicaciones dependiendo de algunos escenarios, por ejemplo:

- **Aplicación (servidor) reiniciando**: El período de las 24:00 horas ya no sería válido.
- **Cambio de horario**: Dependiendo del cambio de horario, tendríamos 1 hora más o menos, es decir, una preocupación adicional. ¿Y ahora?, ¿hay alguna forma de evitar esta situación?.

Programación con Cron

Teniendo en cuenta todos estos escenarios, lo ideal sería hacer uso de algún tipo de recurso que de hecho te permitiera configurar de forma más específica los minutos, horas o incluso el día en que se ejecutará la programación determinada.

Uno de los mecanismos que se utilizan en los sistemas basados en Unix (como es el caso de Linux), para la programación tareas de forma más puntual, es Cron.

Básicamente, Cron es un programa que nos permite ejecutar trabajos en un período determinado, como: cada A segundos, B minutos, C horas en el día D, ¡ejecute una tarea!.

Todos los ajustes se realizan a través de una función llamada **Cron Expression**.

Ejemplos de uso de Cron

A continuación, se muestran algunos ejemplos de ejecución de Cron utilizando Cron Expression:

Cada segundo: ```" 1 "

Todos los días que el horario tenga al menos 01 segundos: ```" 1 "

Cada hora, según el reloj, que esté con 00 minutos y segundos todos los días: ```" 0 0 "

```
Ejemplos: 00:00:00, 01:00:00...
```

Significado de cada campo de Cron

Al principio, Cron puede parecer muy raro, sin embargo, cada campo tiene un significado muy específico. A continuación se enumera lo que significa cada campo junto con sus valores esperados:

```
A B C D E F
```

- **A**: Segundos (0 - 59).
- **B**: Minutos (0 - 59).
- **C**: Horas (0 - 23).
- **D**: Día (1 - 31).
- **E**: Mes (1 - 12).
- **F**: Día de la semana (0 - 6).

También ten en cuenta que en los ejemplos, el (*), este carácter indica que para el campo especificado se considerará cualquier valor.

Observación: Cuando hacemos uso de "/" concatenando algún valor, indicamos lo siguiente: para cada (/)X (valor numérico) repetición active, en otras palabras, */1 indica que si algún valor del campo **segundo** cambia se activará, así que si fuera* /2, cada 2 veces el campo **segundo** cambie de valor, se activará.

Usando Cron en Scheduled

Conociendo todo el poder de Cron en esta pequeña demostración, sería magnífico usar una función similar a Cron en `@Programado`.

Afortunadamente también podemos usarlo a través del parámetro `cron`, es decir, para ejecutar esta programación que obtendría las ventas todos los días al medio día, se podría hacer de la siguiente manera:

```
@Component @EnableScheduling
public class VerificadorDePagos {

  private final long SEGUNDO = 1000;
  private final long MINUTO = SEGUNDO * 60;
  private final long HORA = MINUTO * 60;

  @Scheduled(cron = "0 0 12")
  public void verificaPorHora() {
    System.out.println(LocalDateTime.now());
    *// Código que realiza la consulta de flujo de ventas*
    }
}
```

¡En esta Cron Expression tenemos una ejecución de todos los días al mediodía!

Precauciones al usar Cron

Aunque Cron resuelve nuestro problema de ejecutar una tarea en un período muy específico, existe una situación que puede resultar problemática, que es Time Zone. En

otras palabras, si la ubicación donde está alojado su software tiene una Time Zone diferente a la suya, ¡el horario medio día puede no ser el mismo que el suyo!

De esa manera, Cron causaría más problemas de los que solucionaría, ¿y ahora?

Como siempre vemos, siempre existe esa flexibilidad en configuraciones cuando usamos Spring, ¡y con la programación de tareas no es la excepción!. Es decir, ¡también podemos definir bajo qué Time Zone la `@Scheduled` se ejecutará a través del parámetro `zone`! Entonces se vería así:

```
@Component @EnableScheduling
public class VerificadorDePagos {

    private static final String TIME_ZONE =
  "America/Mexico_City";
    *// atributos*

  @Scheduled(cron = "0 0 12 \* \* \*", zone = TIME_ZONE) public void
  verificaPorHora() {
   System.out.println(LocalDateTime.now());
   *// Código que realiza la consulta de flujo de ventas*
   }
}
```

Mira que ahora estoy enviando el Time Zone `America/Mexico_City` que es precisamente el Time Zone utilizado en Ciudad de México, por lo tanto, la programación se puede realizar sin preocuparse de dónde está asignado el proyecto.

¿Quieres conocer Spring? ¿Qué tal empezar hoy en la Formación de **Desarrollador Java Web con Spring**? En esta selección de cursos, ¡aprenderá a crear una aplicación Web muy robusta desde cero con uno de los frameworks más famosos del mundo Java!. Echa un vistazo también en todo nuestro contenido **Java, Web, Spring, Hibernate, Tests y más**.

Puedes leer también:

- Redondeo de números en Java
- Conozca la API de fechas de Java 8
- Cómo separar palabras de String en Java

ARTÍCULOS DE TECNOLOGÍA > PROGRAMACIÓN

¡QUIERO EMPEZAR A ESTUDIAR!

Paga en moneda local en los siguientes países

ANUAL

US$79,90

un solo pago de US$79,90

- 218 cursos
- Videos y actividades 100% en Español
- Certificado de participación
- Estudia las 24 horas, los 7 días de la semana
- Foro y comunidad exclusiva para resolver tus dudas
- Acceso a todo el contenido de la plataforma por 12 meses

¡CONTÁCTANOS!

¡QUIERO ENTRAR EN CONTACTO!

BLOG

PROGRAMACIÓN
FRONT END
DATA SCIENCE
INNOVACIÓN Y GESTIÓN
DEVOPS

AOVS Sistemas de Informática S.A
CNPJ 05.555.382/0001-33

SÍGUENOS EN NUESTRAS REDES SOCIALES

ALIADOS

Empresa participante do SCALE UP ENDEAVOR

En Alura somos unas de las Scale-Ups seleccionadas por Endeavor, programa de aceleración de las empresas que más crecen en el país.

Google for Startups Alumni 2021

Fuimos unas de las 7 startups seleccionadas por Google For Startups en participar del programa Growth Academy en 2021

POWERED BY

CURSOS

Cursos de Programación
Lógica de Programación | Java

Cursos de Front End
HTML y CSS | JavaScript | React

Cursos de Data Science
Data Science | Machine Learning | Excel | Base de Datos | Data Visualization | Estadística

Cursos de DevOps
Docker | Linux

Cursos de Innovación y Gestión
Productividad y Calidad de Vida | Transformación Ágil | Marketing Analytics | Liderazgo y Gestión de Equipos | Startups y Emprendimiento